DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Subject Name: ARTIFICIAL INTELLIGENCE

Subject Code: CS T71

UNIT – I

Introduction: History of AI – problem spaces and search- Heuristic Search techniques –Best-first search- Problem reduction-Constraint satisfaction-Means Ends Analysis. Intelligent agents: Agents and environment – structure of agents and its functions

Artificial Intelligence

Artificial intelligence is the study of how to make computers do things which, at moment people do better. Artificial intelligence can be viewed from a variety of perspectives.

- ✓ From the perspective of intelligence, artificial intelligence is making machines "intelligent" -- acting as we would expect people to act.
  - The inability to distinguish computer responses from human responses is called the Turing test.
  - Intelligence requires knowledge.

- ✓ From a business perspective AI is a set of very powerful tools, and methodologies for using those tools to solve business problems.

- ✓ From a programming perspective, AI includes the study of symbolic programming, problem solving, and search.
  - Typically AI programs focus on symbols rather than numeric processing.
  - Problem solving i.e. to achieve a specific goal.
  - Search - rarely access a solution directly. Search may include a variety of techniques.

- ✓ It is the science and engineering of making intelligent machines, especially intelligent computer programs.

Artificial Intelligence (AI) is a branch of *Science* which deals with helping machines finding solutions to complex problems in a more human-like fashion. This generally involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way. A more or less flexible or efficient approach can be taken depending on the requirements established, which influences how artificial the intelligent behaviour appears. AI is generally associated with *Computer Science*, but it has many important links with other fields such as *Maths*, *Psychology*, *Cognition*, *Biology* and *Philosophy*, among many others. Our ability to combine knowledge from all these fields will ultimately benefit our progress in the quest of creating an intelligent artificial being.

AI currently encompasses a huge variety of subfields, from general-purpose areas such as perception and logical reasoning, to specific tasks such as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases. Often, scientists in other fields move gradually into artificial intelligence, where they find the tools and vocabulary to systematize and automate the intellectual tasks on which they have been working all their lives. Similarly, workers in AI can choose to apply their methods to any area of human intellectual endeavour. In this sense, it is truly a universal field.

AI Problems

- ✓ Much of the early work in the field of AI focused on formal tasks, such as game playing and theorem proving.
- ✓ Game playing and theorem proving share the property that people who do them well are considered to be displaying Intelligence.
- ✓ Initially computers could perform well at those tasks simply by being fast at exploring a large number of solution paths and then selecting the best one.
- ✓ Humans learn mundane (ordinary) tasks since their birth. They learn by perception, speaking, using language, and training. They learn Formal Tasks and Expert Tasks later.
- ✓ Another early foray into AI focused on commonsense reasoning, which includes reasoning about physical objects and their relationship to each other, as well as reasoning about actions and their consequences.
- ✓ As AI research progressed, techniques for handling large amount of world knowledge were developed.
- ✓ New tasks reasonably attempted such as perception, natural language understanding and problem solving in specialized domains.
- ✓ Some of the task domains of artificial intelligence are presented in table I.
- ✓ Earlier, all work of AI was concentrated in the mundane task domain.

Mundane tasks	Formal tasks	Expert tasks
Perception – Computer Vision – Speech, Voice	Games – Go – Chess (Deep Blue) – Ckeckers	Engineering – Design – Fault Finding – Manufacturing – Monitoring
Natural Language Processing – Understanding – Language Generation – Language Translation	Mathematics – Geometry – Logic – Integration and Differentiation	Scientific Analysis
Common Sense Reasoning	Theorem Proving	Financial Analysis
Planning		Medical Diagnosis
Robot Control		

Table I Task Domains of AI

Later, it turned out that the machine requires more knowledge, complex knowledge representation, and complicated algorithms for handling mundane tasks.

- ✓ This is the reason why AI work is more flourishing in the Expert Tasks domain now, as the expert task domain needs expert knowledge without common sense, which can be easier to represent and handle.

What is an AI technique?

- ✓ Artificial intelligence problems span a very broad spectrum. They appear to have very little in common except that they are hard.
- ✓ AI Research of earlier decades results into the fact that intelligence requires knowledge.
- ✓ Knowledge possess following properties:
  - It is voluminous.
  - It is not well-organized or well-formatted.
  - It is constantly changing.
  - It differs from data. And it is organized in a way that corresponds to its usage.

- ✓ AI technique is a method that exploits knowledge that should be represented in such a way that:
  - Knowledge captures generalization. Situations that share common properties are grouped together. Without this property, inordinate amount of memory and modifications will be required.
  - It can be understood by people who must provide it. Although bulk of data can be acquired automatically, in many AI domains most of the knowledge must ultimately be provided by people in terms they understand.
  - It can easily be modified to correct errors and to reflect changes in the world.
  - It can be used in many situations even though it may not be totally accurate or complete.
  - It can be used to reduce its own volume by narrowing range of possibilities.

- ✓ There are three important AI techniques:

1. Search –
   - Provides a way of solving problems for which no direct approach is available.
   - b. It also provides a framework into which any direct techniques that are available can be embedded.

2. Use of knowledge – Provides a way of solving complex problems by exploiting the structure of the objects that are involved.

3. Abstraction –Provides a way of separating important features and variations from many unimportant ones that would otherwise overwhelm any process.

Classification of AI

1. Weak AI: The study and design of machines that perform intelligent tasks.
- ✓ Not concerned with how tasks are performed, mostly concerned with performance and efficiency, such as solutions that are reasonable for NP-Complete problems. E.g., to make a flying machine, use logic and physics, don't mimic a bird.

2. Strong AI: The study and design of machines that simulate the human mind to perform intelligent tasks.

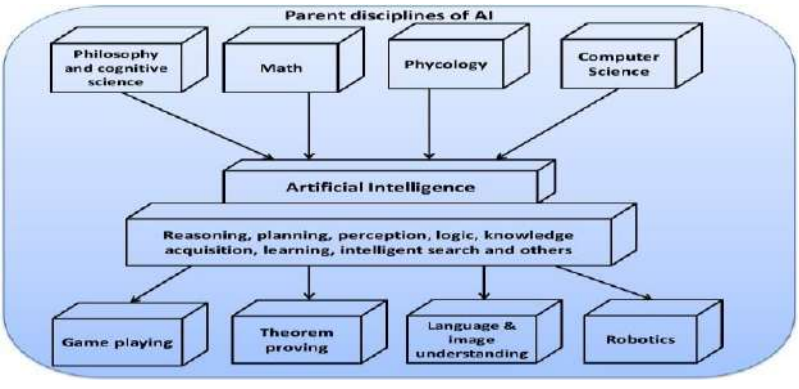
- Borrow many ideas from psychology, neuroscience. Goal is to perform tasks the way a human might do them – which makes sense, since we do have models of human thought and problem solving.
- Includes psychological ideas in STM, LTM, forgetting, language, genetics, etc. Assumes that the physical symbol hypothesis holds.

3. Evolutionary AI. The study and design of machines that simulate simple creatures, and attempt to evolve and have higher level emergent behavior. For example, ants, bees, etc.

Applications of AI

AI has been dominant in various fields such as −

1. Gaming − AI plays vital role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
2. Natural Language Processing − It is possible to interact with the computer that understands natural language spoken by humans.
3. Expert Systems − There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
4. Computer Vision Systems − These systems understand, interpret, and comprehend visual input on the computer.
5. Speech Recognition − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise, etc.
6. Handwriting Recognition − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.
7. Intelligent Robots − Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment



COMPONENTS OF AI - There are three types of components in AI

1) Hardware Components of AI

a) Pattern Matching

b) Logic Representation
c) Symbolic Processing
d) Numeric Processing
e) Problem Solving
f) Heuristic Search
g) Natural Language processing
h) Knowledge Representation
i) Expert System
j) Neural Network
k) Learning
l) Planning
m) Semantic Network

2) Software Components
a) Machine Language
b) Assembly language
c) High level Language
d) LISP Language
e) Fourth generation Language
f) Object Oriented Language
g) Distributed Language
h) Natural Language
i) Particular Problem Solving Language

3) Architectural Components
a) Uniprocessor
b) Multiprocessor
c) Special Purpose Processor
d) Array Processor
e) Vector Processor
f) Parallel Processor
g) Distributed Processor

Problems, State Space Search & Heuristic Search Techniques

- ✓ Problem solving is the major area of concern in Artificial Intelligence.
- ✓ It is the process of generating solution from given observed data.
- ✓ To solve a particular problem, we need to build a system or a method which can generate required solution.

Following four things are required for building such system.

1. Define the problem precisely.
- ✓ This definition must precisely specify the initial situation (input).
- ✓ What final situation (output) will constitute the acceptable solution to the problem?

2. Analyze the problem.
- ✓ To identify those important features which can have an immense impact on the appropriateness of various possible techniques for solving the problem.

3. Isolate and represent the task knowledge that is necessary to solve the problem.

4. Choose the best problem solving technique and apply it to the particular problem.

Defining the Problem as a State Space Search

1. Defining Problem & Search

- ✓ A problem is described formally as:
  - Define a state space that contains all the possible configurations of relevant objects.
  - Specify one or more states within that space that describe possible situations from which the problem solving process may start. These states are called initial states.
  - Specify one or more states that would be acceptable as solutions to the problem. These states are called goal states.
  - Specify a set of rules that describe the actions available.

- ✓ The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found.
  - This process is known as search.
  - Search is fundamental to the problem-solving process.
  - Search is a general mechanism that can be used when more direct method is not known.
  - Search also provides the framework into which more direct methods for solving subparts of a problem can be embedded.

2. Defining State & State Space

- ✓ A state is a representation of problem elements at a given moment.
- ✓ A State space is the set of all states reachable from the initial state.
- ✓ A state space forms a graph in which the nodes are states and the arcs between nodes are actions.
- ✓ In state space, a path is a sequence of states connected by a sequence of actions.
- ✓ The solution of a problem is part of the graph formed by the state space.
- ✓ The state space representation forms the basis of most of the AI methods.
- ✓ Its structure corresponds to the structure of problem solving in two important ways:

1. It allows for a formal definition of a problem as per the need to convert some given situation into some desired situation using a set of permissible operations.
2. It permits the problem to be solved with the help of known techniques and control strategies to move through the problem space until goal state is found.

3. Define the Problem as State Space Search

Problems dealt with in artificial intelligence generally use a common term called 'state'. A state represents a status of the solution at a given step of the problem solving procedure. The solution of a problem, thus, is a collection of the problem states. The problem solving procedure applies an operator to a state to get the next state. Then it applies another operator to the resulting state to derive a new state. The process of applying an operator to a state and its subsequent transition to the next state, thus, is continued until the goal (desired) state is derived. Such a method of solving a problem is generally referred to as **state space approach**

Ex.1:- Consider the problem of Playing Chess

- To build a program that could play chess, we have to specify:
  - The starting position of the chess board,
  - The rules that define legal moves, and
  - The board position that represents a win.

- The starting position can be described by an 8 X 8 array square in which each element square (x, y), (x varying from 1 to 8 & y varying from 1 to 8) describes the board position of an appropriate piece in the official chess opening position.

- The goal is any board position in which the opponent does not have a legal move and his or her "king" is under attack.

- The legal moves provide the way of getting from initial state of final state.

- The legal moves can be described as a set of rules consisting of two parts: A left side that gives the current position and the right side that describes the change to be made to the board position.
- An example is shown in the following figure.

Current Position

While pawn at square (5 , 2), AND Square (5 , 3) is empty, AND Square (5 , 4) is empty.

Changing Board Position

Move pawn from Square (5 , 2) to Square (5 , 4).

- ✓ The current position of a chess coin on the board is its state and the set of all possible states is state space.

- ✓ One or more states where the problem terminates are goal states.

- ✓ Chess has approximately 10^{120} game paths. These positions comprise the problem search space. Using above formulation, the problem of playing chess is defined as a problem of moving around in a state space, where each state corresponds to a legal position of the board.

- ✓ State space representation seems natural for play chess problem because the set of states, which corresponds to the set of board positions, is well organized.

Ex.2:- Water Jug problem

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

- ✓ Here the initial state is (0, 0). The goal state is (2, n) for any value of n.

- ✓ **State Space Representation:** we will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug. Note that $0 \leq x \leq 4$, and $0 \leq y \leq 3$.
- ✓ To solve this we have to make some assumptions not mentioned in the problem. They are:
  - We can fill a jug from the pump.
  - We can pour water out of a jug to the ground.
  - We can pour water from one jug to another.
  - There is no measuring device available.
- ✓ Operators – we must define a set of operators that will take us from one state to another.

Rule No	Production Rule	Action
1	(i, j) ——— (4,j) if i<4.	Fill the 4-liter jug, if 4-liter jug is not full.
2	(i, j) ——— (i,3) if j<3.	Fill the 3-liter jug, if 3-liter jug is not full.
3	(i, j) ——— (i-s,j) if i>0.	Pour some water our of the 4-liter jug, if 4-liter jug is not empty.
4	(i, j) ——— (i,j-s) if j>0.	Pour some water out of the 3-liter jug, if 3-liter jug is not empty.
5	(i, j) ——— (0,j) if i>0.	Empty the 4-liter jug on the ground, if 4-liter jug is not empty.
6	(i, j) ——— (i,0) if j>0.	Empty the 3-liter jug on the ground, if 3-liter jug is not empty.
7	(i, j) ——— (4,j-(4-i)) if (i+j)>=4 & j>0.	Pour water from the 30-liter jug into the 4-liter jug until the 4 liter jug is full, if the combined content is >= 4 and 3-liter jug is not empty.
8	(i, j) ——— (i,(3-j),3) if (i+j) >= 3 & i>0.	Pour water from the 4-liter the into the 3 liter jug until the 3-liter jug is full if the combined content is >=3 and 4-liter jug is not empty.
9	(i, j) ——— (i+j,0) if (i+j)<=4 and j>0.	Pour all the water from the 3-liter jug into the 4-liter jug if the jug, combined content is <=4 and 3-liter jug is not empty.
10	(i, j) ——— (0,i+j) if i+j)<=3 and i>0	Pour all the water from the 4-liter jug into the3-liter jug, if the combined content is <=3 and 4-liter jug is not empty

For the water jug problem , there are several sequence of operators that will solve the problem . let us see of them.

SOLUTION 1;-

Liters in the 4-liter jug	Liters in the 3-liter jug	Rule applied
0	0	
4	0	1
1	3	8
1	0	6
0	1	10
4	1	1
2	3	8

Solution 2:-

Liters in the 4-liter jug	Liters in the 3-liter jug	Rule applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5
2	0	9

Solution 3:-

Liters in the 4-liter jug	Liters in the 3-liter jug	Rule applied
0	0	
4	0	1
1	3	8
0	3	5
3	0	9
3	3	2
4	2	7
0	2	5
2	0	9

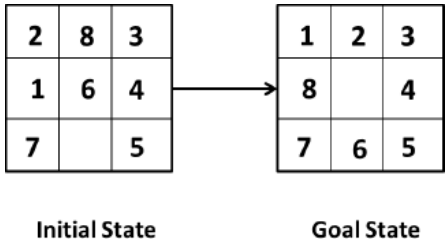
Figures gives comparative study of the above 3 different solutions.

Fig :- comparison of 3 solutions.

We see that, when there is no limit for water prevails then solution is the most efficient. When water is limited then Solution2 is the best suited. In no way, solution 3 is good, Because it it requires 8 steps to solution and wastes 5 liters of water.

Ex.3:- Consider 8 puzzle problem

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following diagram.



- The program is to change the initial configuration into the goal configuration.
- A solution to the problem is an appropriate sequence of moves, such as "move tiles 5 to the right, move tile 7 to the left ,move tile 6 to the down" etc…
- To solve a problem, we must specify the global database, the rules, and the control strategy.
- For the 8 puzzle problem that correspond to three components.
- These elements are the problem states, moves and goal.
- In this problem each tile configuration is a state.
- The set of all possible configuration in the problem space, consists of 3,62,880 different configurations of the 8 tiles and blank space.
- For the 8-puzzle, a straight forward description is a 3X3 array of matrix of numbers.
- Initial global database is this description of the initial problem state. Virtually any kind of data structure can be used to describe states.
- A move transforms one problem state into another state.

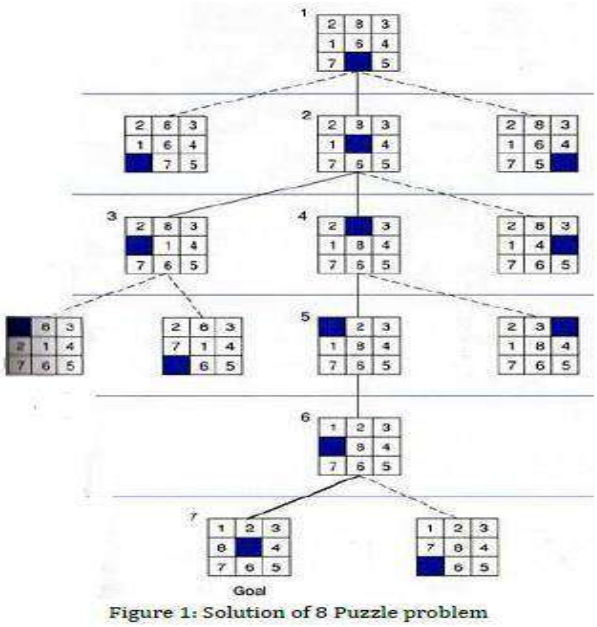


Figure 1: Solution of 8 Puzzle problem

- The 8-puzzle is conveniently interpreted as having the following for moves.

1. Move empty space (blank) to the left,

2. move blank up,
3. Move blank to the right and
4. Move blank down.

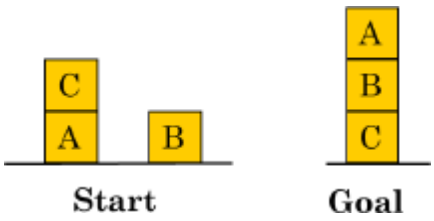
- These moves are modeled by production rules that operate on the state descriptions in the appropriate manner.
- The goal condition forms the basis for the termination.
- The control strategy repeatedly applies rules to state descriptions until a description of a goal state is produced.
- It also keeps track of rules that have been applied so that it can compose them into sequence representing the problem solution.

Problem Characteristics

- In order to choose the most appropriate problem solving method, it is necessary to analyze the problem along various key dimensions.
- These dimensions are referred to as problem characteristics discussed below.

1. Is the problem decomposable into a set of independent smaller or easier sub-problems?

- A very large and composite problem can be easily solved if it can be broken into smaller problems and recursion could be used.
- For example, we want to solve :- $\int x^2 + 3x + \sin 2x \cos 2x \, dx$
- This can be done by breaking it into three smaller problems and solving each by applying specific rules. Adding the results we can find the complete solution.
- But there are certain problems which cannot be decomposed into sub-problems.
- For example Blocks world problem in which, start and goal state are given as,



- Here, solution can be achieved be moving blocks in a sequence such that goal state can be derived.
- Solution steps are interdependent and cannot be decomposed in sub problems.
- These two examples, symbolic integration and the blocks world illustrate the difference between decomposable and non-decomposable problems.

2. Can solution steps be ignored or at least undone if they prove unwise?

- Problem fall under three classes, (i) ignorable, (ii) recoverable and (iii) irrecoverable.
- This classification is with reference to the steps of the solution to a problem.

- Consider theorem proving. We may later find that it is of no use. We can still proceed further, since nothing is lost by this redundant step. This is an example of ignorable solutions steps.

- Now consider the 8 puzzle problem tray and arranged in specified order.

- While moving from the start state towards goal state, we may make some stupid move but we can backtrack and undo the unwanted move. This only involves additional steps and the solution steps are recoverable.

- Lastly consider the game of chess.
  - If a wrong move is made, it can neither be ignored nor be recovered. The thing to do is to make the best use of current situation and proceed. This is an example of an irrecoverable solution steps.

- Knowledge of these will help in determining the control structure.
  - Ignorable problems can be solved using a simple control structure that never backtracks.
  - Recoverable problems can be solved by a slightly more complicated control strategy that allows backtracking.
  - Irrecoverable problems will need to be solved by a system that expends a great deal of effort making each decision since decision must be final.

3. Is the problem's universe predictable?

- Problems can be classified into those with certain outcome (eight puzzle and water jug problems) and those with uncertain outcome (playing cards).
- In certain – outcome problems, planning could be done to generate a sequence of operators that guarantees to lead to a solution.
- Planning helps to avoid unwanted solution steps.
- For uncertain outcome problems, planning can at best generate a sequence of operators that has a good probability of leading to a solution.
- The uncertain outcome problems do not guarantee a solution and it is often very expensive since the number of solution paths to be explored increases exponentially with the number of points at which the outcome cannot be predicted.
- Thus one of the hardest types of problems to solve is the irrecoverable, uncertain – outcome problems (Ex:- Playing cards).

4. Is a good solution to the problem obvious without comparison to all other possible solutions?

- There are two categories of problems - Any path problem and Best path problem.
  - In any path problem, like the water jug and 8 puzzle problems, we are satisfied with the solution, irrespective of the solution path taken.
  - Whereas in the other category not just any solution is acceptable but we want the best path solution.
    - Like that of traveling sales man problem, which is the shortest path problem.
- In any – path problems, by heuristic methods we obtain a solution and we do not explore alternatives.
- Any path problems can often be solved in a reasonable amount of time by using heuristics that suggest good paths to explore.
- For the best-path problems all possible paths are explored using an exhaustive search until the best path is obtained.
- Best path problems are computationally harder.

5. Is the desired solution a state of the world or a path to a state?

- Consider the problem of natural language processing.

- Finding a consistent interpretation for the sentence "The bank president ate a dish of pasta salad with the fork".

- We need to find the interpretation but not the record of the processing by which the interpretation is found.
- Contrast this with the water jug problem.In water jug problem, it is not sufficient to report that we have solved, but the path that we found to the state (2, 0). Thus the statement of a solution to this problem must be a sequence of operations that produces the final state.

6. What is the role of knowledge?

- Though one could have unlimited computing power, the size of the knowledge base available for solving the problem does matter in arriving at a good solution.
- Take for example the game of playing chess, just the rules for determining legal moves and some simple control mechanism is sufficient to arrive at a solution.
- But additional knowledge about good strategy and tactics could help to constrain the search and speed up the execution of the program. The solution would then be realistic.
- Consider the case of predicting the political trend. This would require an enormous amount of knowledge even to be able to recognize a solution, leave alone the best.

7. Does the task require interaction with a person?

The problems can again be categorized under two heads.

1. Solitary in which the computer will be given a problem description and will produce an answer, with no intermediate communication and with the demand for an explanation of the reasoning process. Simple theorem proving falls under this category. Given the basic rules and laws, the theorem could be proved, if one exists.

2. Conversational, in which there will be intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user, or both, such as medical diagnosis fall under this category, where people will be unwilling to accept the verdict of the program, if they cannot follow its reasoning.

Problem Classification

Actual problems are examined from the point of view of all these questions; it becomes apparent that there are several broad classes into which the problems fall.

Issues in the design of search programs

1. The direction in which to conduct the search (forward versus backward reasoning). If the search proceeds from start state towards a goal state, it is a forward search or we can also search from the goal.
2. How to select applicable rules (Matching). Production systems typically spend most of their time looking for rules to apply. So, it is critical to have efficient procedures for matching rules against states.
3. How to represent each node of the search process (knowledge representation problem).

Production System

The production system is a model of computation that can be applied to implement search algorithms and model human problem solving. Such problem solving knowledge can be packed up in the form of little quanta called productions.

A production is a rule consisting of a situation recognition part and an action part. A production is a situation-action pair in which the left side is a list of things to watch for and the right side is a list of things to do so.

When productions are used in deductive systems, the situation that trigger productions are specified combination of facts. The actions are restricted to being assertion of new facts deduced directly from the triggering combination. Production systems may be called premise conclusion pairs rather than situation action pair.

- Search process forms the core of many intelligence processes.
- So, it is useful to structure AI programs in a way that facilitates describing and performing the search process.
- Production system provides such structures.

A production system consists of:

1. A set of rules, each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed if that rule is applied.

2. One or more knowledge/databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem.

3. A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.

4. A rule applier which is the computational system that implements the control strategy and applies the rules.

✓ In order to solve a problem:
- We must first reduce it to the form for which a precise statement can be given. This can be done by defining the problem's state space (start and goal states) and a set of operators for moving that space.
- The problem can then be solved by searching for a path through the space from an initial state to a goal state.
- The process of solving the problem can usefully be modeled as a production system.

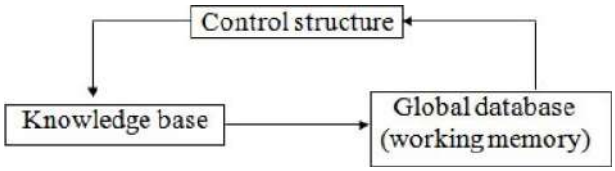
The important roles played by production systems include a powerful knowledge representation scheme. A production system not only represents knowledge but also action. It acts as a bridge between AI and expert systems. Production system provides a language in which the representation of expert knowledge is very natural. We can represent knowledge in a production system as a set of rules of the form

If (condition) THEN (condition)

along with a control system and a database. The control system serves as a rule interpreter and sequencer. The database acts as a context buffer, which records the conditions evaluated by the rules and information on which the rules act. The production rules are also known as condition – action, antecedent – consequent, pattern – action, situation – response, feedback – result pairs.

For example,

If (you have an exam tomorrow)
THEN (study the whole night)



Architecture of Production System

Features of Production System
Some of the main features of production system are:

Expressiveness and intuitiveness: In real world, many times situation comes like "if this happen-you will do that", "if this is so-then this should happen" and many more. The production rules essentially tell us what to do in a given situation.

1. **Simplicity:** The structure of each sentence in a production system is unique and uniform as they use "IF-THEN" structure. This structure provides simplicity in knowledge representation. This feature of production system improves the readability of production rules.

2. **Modularity:** This means production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deletetious side effects.

3. **Modifiability:** This means the facility of modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.

4. **Knowledge intensive:** The knowledge base of production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

Disadvantages of production system
1. **Opacity:** This problem is generated by the combination of production rules. The opacity is generated because of less prioritization of rules. More priority to a rule has the less opacity.

2. **Inefficiency:** During execution of a program several rules may active. A well devised control strategy reduces this problem. As the rules of the production system are large in number and they are hardly written in hierarchical manner, it requires some forms of complex search through all the production rules for each cycle of control program.

3. **Absence of learning:** Rule based production systems do not store the result of the problem for future use. Hence, it does not exhibit any type of learning capabilities. So for each time for a particular problem, some new solutions may come.

4. **Conflict resolution:** The rules in a production system should not have any type of conflict operations. When a new rule is added to a database, it should ensure that it does not have any conflicts with the existing rules.

Benefits of Production System

- Production systems provide an excellent tool for structuring AI programs.
- Production Systems are highly modular because the individual rules can be added, removed or modified independently.
- The production rules are expressed in a natural form, so the statements contained in the knowledge base should be easily understandable.

Production System Characteristics

The production system can be classified as monotonic, non-monotonic, partially commutative and commutative.

1. **Monotonic Production System:** the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected. i.e., rules are independent.

   Monotonic learning is when an agent may not learn any knowledge that contradicts what it already knows. For example, it may not replace a statement with its negation. Thus, the knowledge base may only grow with new facts in a monotonic fashion. The advantages of monotonic learning are:
   1. Greatly simplified truth-maintenance
   2. Greater choice in learning strategies

2. **Non-Monotonic Production system - Non-monotonic learning** is when an agent may learn knowledge that contradicts what it already knows. So it may replace old knowledge with new if it believes there is sufficient reason to do so. The advantages of non-monotonic learning are:
   1. increased applicability to real domains,
   2. greater freedom in the order things are learned in

   A related property is the consistency of the knowledge. If an architecture must maintain a consistent knowledge base then any learning strategy it uses must be monotonic.

3. **Partially commutative Production system:** a production system with the property that if application of a particular sequence of rules transforms state x to state y, then allowable permutation of those rules, also transforms state x into state y.

4. **Commutative Production system**: A Commutative production system is a production system that is both monotonic and partially commutative.

Control Strategies

- Control strategies help us decide which rule to apply next during the process of searching for a solution to a problem.

Problem solving in artificial intelligence may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. In AI problem solving by search algorithms is quite common technique. In the coming age of AI it will have big impact on the technologies of the robotics and path finding. It is also widely used in travel planning. A search algorithm takes a problem as input and returns the solution in the form of an action sequence. Once the solution is found, the actions it recommends can be carried out. This phase is called as the execution phase. After formulating a goal and problem to solve the agent cells a search procedure to solve it. A problem can be defined by 5 components.

a) **The initial state**: The state from which agent will start.
b) **The goal state**: The state to be finally reached.
c) **The current state**: The state at which the agent is present after starting from the initial state.
d) **Successor function**: It is the description of possible actions and their outcomes.
e) **Path cost**: It is a function that assigns a numeric cost to each path.

- Good control strategy should:
    1. It should cause motion
    2. It should be Systematic

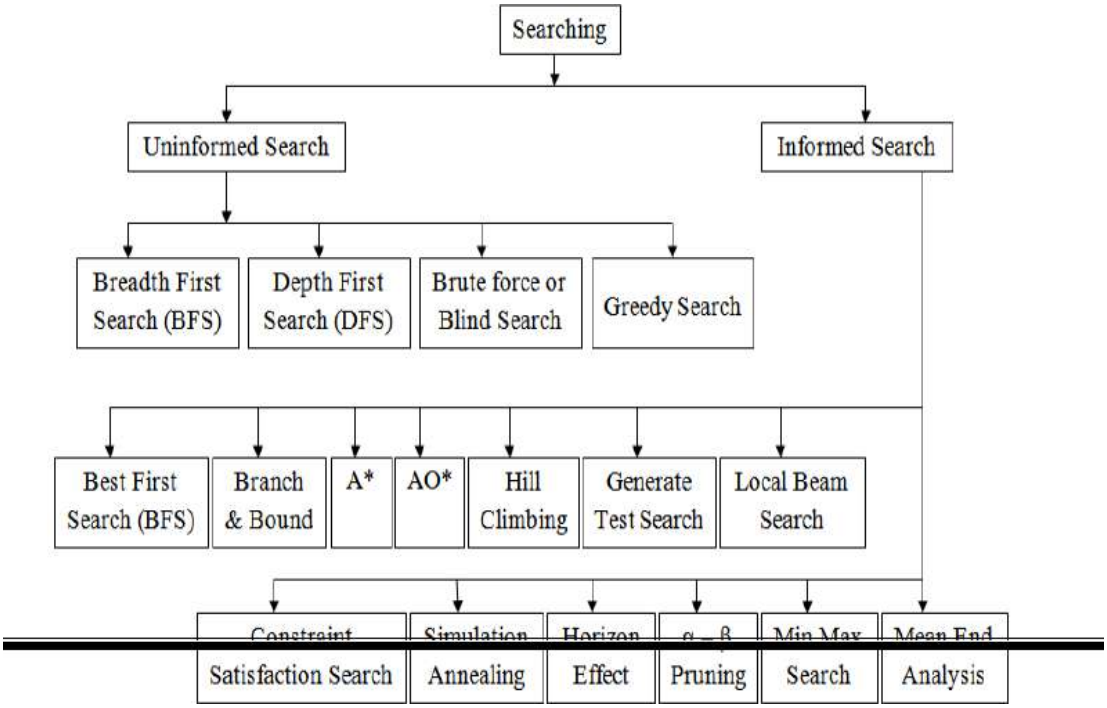
- Control strategies are classified as:

    1. **Uninformed/blind search control strategy:**
        - Do not have additional information about states beyond problem definition.
        - Total search space is looked for solution.

        Example: Breadth First Search (BFS), Depth First Search (DFS), Depth Limited Search (DLS).

    2. **Informed/Directed Search Control Strategy:**
        - Some information about problem space is used to compute preference among the various possibilities for exploration and expansion.

    Examples: Best First Search, Problem Decomposition, A*, Mean end Analysis



Uninformed Search

Breadth First Search (BFS)

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. In this type of search the state space is represented in form of a tree. The solution is obtained by traversing through the tree. The nodes of the tree represent the start value or starting state, various intermediate states and the final state. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution if one exists. The solution which is found is always the optional solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level.

Concept:

Step 1: Traverse the root node
Step 2: Traverse all neighbours of root node.
Step 3: Traverse all neighbours of neighbours of the root node.
Step 4: This process will continue until we are getting the goal node.

Algorithm:

1. Create a variable called NODE-LIST and set it to initial state.
2. Until a goal state is found or NODE-LIST is empty do:
    i. Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit.
    ii. For each way that each rule can match the state described in E do:
        a. Apply the rule to generate a new state.
        b. If the new state is a goal state, quit and return this state.
        c. Otherwise, add the new state to the end of NODE-LIST.

Depth First Search (DFS)

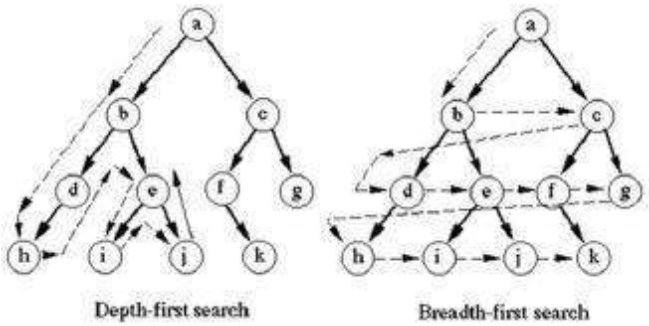
DFS is also an important type of uninformed search. DFS visits all the vertices in the graph. This type of algorithm always chooses to go deeper into the graph. After DFS visited all the reachable vertices from a particular sources vertices it chooses one of the remaining undiscovered vertices and continues the search. DFS reminds the space limitation of breath first search by always generating next a child of the deepest unexpanded nodded. The data structure stack or last in first out (LIFO) is used for DFS. One interesting property of DFS is that, discover and finish time of each vertex from a parenthesis structure. If we use one open parenthesis when a vertex is finished then the result is properly nested set of parenthesis.

Concept:
Step 1: Traverse the root node.
Step 2: Traverse any neighbour of the root node.
Step 3: Traverse any neighbour of neighbour of the root node.
Step 4: This process will continue until we are getting the goal node.

Algorithm:
1. If the initial state is a goal state, quit and return success
2. Otherwise, do the following until success or failure is signaled:
   a. Generate a successor, E, of initial state. If there are no more successors, signal failure.
   b. Call Depth-First Search, with E as the initial state
   c. If success is returned, signal success. Otherwise continue in this loop.

Comparison: DFS & BFS

Depth First Search	Breath First Search
DFS requires less memory since only the nodes on the current path are stored.	BFS guarantees that the space of possible moves is systematically examined; this search requires considerable memory resources.
By chance, DFS may find a solution without examining much of the search space at all. Then it finds solution faster.	The search systematically proceeds testing each node that is reachable from a parent node before it expands to any child of those nodes.
If the selected path does not reach to the solution node, DFS gets stuck into a blind alley.	BFS will not get trapped exploring a blind alley.
Does not guarantee to find solution. Backtracking is required if wrong path is selected.	If there is a solution, BFS is guaranteed to find it.



Depth-first search

Breadth-first search

Brute Force or Blind Search

Brute force or blind search is a uniformed exploration of the search space and it does not explicitly take into account either planning efficiency or execution efficiency. Blind search is also called uniform search. It is the search which has no information about its domain. The only thing that a blind search can do is to differentiate between a non-goal state and a goal state. These methods do not need domain knowledge but they are less efficient in result. Uniform strategies don't use any information about how a close a node might be to a goal. They differ in the order that the nodes are expanded. The most important brute force techniques are breadth first search, depth first search, uniform search and bidirectional search. All brute force techniques must take (b^d time and use o (d) space. This technique is not as efficient as compared to other algorithms.

Greedy Search

This algorithm uses an approach which is quite similar to the best first search algorithm. It is a simple best first search which reduces the estimated cost of reach the goal. Basically it takes the closest node that appears to be closest to the goal. This search starts with the initial matrix and makes very single possible changes then looks at the change it made to the score. This search then applies the change till the greatest improvement. The search continues until no further improvement can be made. The greedy search never makes never makes a lateral move .It uses minimal estimated cost h (n) to the goal state as measure which decreases the search time but the algorithm is neither complete nor optimal. The main advantage of this search is that it is simple and finds solution quickly. The disadvantages are that it is not optimal, susceptible to false start.

Heuristic Search Techniques (Informed Techniques)

- In order to solve many hard problems efficiently, it is often necessary to compromise the requirements of mobility and systematically and to construct a control structure that is no longer guaranteed to find the best answer but will always find a very good answer.
- Usually very hard problems tend to have very large search spaces. Heuristics can be used to limit search process.
- There are good general purpose heuristics that are useful in a wide variety of problem domains.
- Special purpose heuristics exploit domain specific knowledge.
- For example nearest neighbor heuristics for shortest path problem. It works by selecting locally superior alternative at each step.
- Applying nearest neighbor heuristics to Travelling Salesman Problem:

1. Arbitrarily select a starting city
2. To select the next city, look at all cities not yet visited and select the one closest to the current city. Go to next step.
3. Repeat step 2 until all cities have been visited.

This procedure executes in time proportional to N^2, where N is the number of cities to be visited.

Heuristic Function

- Heuristic is a problem specific knowledge that decreases expected search efforts. It is a technique which sometimes works but not always.
- Heuristic search algorithm uses information about the problem to help directing the path through the search space. These searches uses some functions that estimate the cost from the current state to the goal presuming that such function is efficient.
- A heuristic function is a function that maps from problem state descriptions to measure of desirability usually represented as number. The purpose of heuristic function is to guide the search process in the most profitable directions by suggesting which path to follow first when more than is available
- Well-designed heuristic functions can provides a fairly good estimate of whether a path is good or not. (" The sum of the distances traveled so far" is a simple heuristic function in the traveling salesman problem)

- Heuristic function maps from problem state descriptions to measures of desirability, usually represented as numbers.
- Which aspects of the problem state are considered, how those aspects are evaluated, and the weights given to individual aspects are chosen in such a way that the value of the heuristic function at a given node in the search process gives as good an estimate as possible of whether that node is on the desired path to a solution.
- Well-designed heuristic functions can play an important part in efficiently guiding a search process toward a solution.
- Every search process can be viewed as a traversal of a directed graph, in which the nodes represent problem states and the arcs represent relationships between states.
- The search process must find a path through this graph, starting at an initial state and ending in one or more final states.
- Domain-specific knowledge must be added to improve search efficiency. Information about the problem includes the nature of states, cost of transforming from one state to another, and characteristics of the goals.
- This information can often be expressed in the form of heuristic evaluation function.
- In general, heuristic search improve the quality of the path that are exported.
- Using good heuristics we can hope to get good solutions to hard problems such as the traveling salesman problem in less than exponential time.

A Heuristic technique helps in solving problems, even though there is no guarantee that it will never lead in the wrong direction. There are heuristics of every general applicability as well as domain specific. The strategies are general purpose heuristics. In order to use them in a specific domain they are coupler with some domain specific heuristics. There are two major ways in which domain - specific, heuristic information can be incorporated into rule-based search procedure.

- In the rules themselves
- As a heuristic function that evaluates individual problem states and determines how desired they are.

HEURISTIC SEARCH

To solve complex problems efficiently, it is necessary to compromise the requirements of the movability and systematically. A control structure has to be constructed that no longer guarantees the best solution, but that will almost always find a very good answer. Such a technique is said to be heuristic (rule of thumb).

A heuristic search improves the efficiently of the search process, but sacrifices the claims of completeness. But they improve the quality of the paths that are explored. Using good heuristics we can get good solutions to hard problems, such as the traveling salesman problem.

Applying it to the traveling salesman problem produces the following procedure.
1. Arbitrarily select a starting city.
2. To select the next city, look at all cities not yet visited. Select the one closet to the current city. Go to it next.
3. Repeat step 2 until all the cities have been visited.

This procedure executes in time proportional to N * N , instead of N! And it is possible to prove an upper bound on the error it incurs. In many AI problems, however, it is not possible to produce such bounds. This is true for two reasons.

For real world problems, it is often hard to measure precisely the goodness of a particular solution. For instance, answers to questions like “Why has inflation increased?” cannot be precise.

ii) For real world problems it is often useful to introduce heuristics based on relatively unstructured knowledge. This is because often a mathematical analysis is not possible. Without heuristics, it is not possible to tackle combinatorial explosion. Moreover, we go for optimum solution that satisfy some set of requirements. We stop with satisfactory solutions even though there might be better solutions.

Ex.A good example of this is the search for a taking space. Most people will stop as soon as there find a fairly good space, even if there must be a slightly better space or ahead.

Heuristic Search Techniques.

Introduction:- Many if the problems are too complex to be solvable by direct techniques. They have to be solved only by suitable heuristic search techniques. Though the heuristic techniques can be described independently, they are domain specific. They are called " Weak Methods", since they are vulnerable to combinatorial explosion. Even so, they provide the frame work into which domain specific knowledge can be placed.

Every search process can be viewed as a traversal of a directed graph, in which the nodes represent problem states and the arcs represent relationships between states. The search process must find a path through this graph, starting at an initial state and ending in one or more final states. The following issues have to be considered before going for a search.

Heuristic Search Techniques.

Heuristic techniques are called weak methods, since they are vulnerable to combinatorial

explosion. Even then these techniques continue to provide framework into which domain specific knowledge can be placed, either by hand or as a result of learning. The following are some general purpose control strategies (often called weak methods).

- Generate - and – test
- Hill climbing
- Best First Search (A* search)
- Problem reduction(AO* search)
- Constraint satisfaction
- Means - ends analysis

A heuristic procedure, or heuristic, is defined as having the following properties.

1. It will usually find good, although not necessary optimum solutions.

2. It is faster and easier to implement than any known exact algorithm
(one which guarantees an optimum solution).

In general, heuristic search improve the quality of the path that are exported. Using good heuristics we can hope to get good solutions to hard problems such as the traveling salesman problem in less than exponential time. There are some good general purpose heuristics that are useful in a wide variety of problems. It is also possible to construct special purpose heuristics to solve particular problems. For example, consider the traveling salesman problem.

Heuristic Search Techniques

I. Generate-and-Test

Generate-and-test search algorithm is a very simple algorithm that guarantees to find a solution if done systematically and there exists a solution.

Algorithm:

1. Generate a possible solution. For some problems, this means generating a particular point in the problem space. For others it means generating a path from a start stat.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit, Otherwise return to step 1.

- It is a depth first search procedure since complete solutions must be generated before they can be tested.
- In its most systematic form, it is simply an exhaustive search of the problem space.
- It operates by generating solutions randomly.

II. Simple Hill Climbing

This is a variety of depth-first (generate - and - test) search. A feedback is used here to decide on the direction of motion in the search space. In the depth-first search, the test function will merely accept or reject a solution. But in hill climbing the test function is provided with a heuristic function which provides an estimate of how close a given state is to goal state. The hill climbing test procedure is as follows:

1. Generate the first proposed solution as done in depth-first procedure. See if it is a solution. If so quit, else continue.
2. From this solution generate new set of solutions use, some application rules
3. For each element of this set

   i. Apply test function. It is a solution quit.
   ii. Else see whether it is closer to the goal state than the solution already generated. If yes, remember it else discard it.

4. Take the best element so far generated and use it as the next proposed solution. This step corresponds to move through the problem space in the direction Towards the goal state.

5. Go back to step 2

- Hill climbing is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available.
- The key difference between Simple Hill climbing and Generate-and-test is the use of evaluation function as a way to inject task specific knowledge into the control process.

Algorithm:

1. Evaluate the initial state. If it is also goal state, then return it and quit. Otherwise continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
   a. Select an operator that has not yet been applied to the current state and apply it to produce a new state.
   b. Evaluate the new state
      i. If it is the goal state, then return it and quit.
      ii. If it is not a goal state but it is better than the current state, then make it the current state.
      iii. If it is not better than the current state, then continue in the loop.

III. Steepest-Ascent Hill Climbing

- This is a variation of simple hill climbing which considers all the moves from the current state and selects the best one as the next state.
- At each current state we select a transition, evaluate the resulting state, and if the resulting state is an improvement we move there, otherwise we try a new transition from where we were.
- We repeat this until we reach a goal state, or have no more transitions to try.
- The transitions explored can be selected at random, or according to some problem specific heuristics.

Algorithm

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   a. Let S be a state such that any possible successor of the current state will be better than S.

b. For each operator that applies to the current state do:
    i. Apply the operator and generate a new state
    ii. Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to S. If it is better, then set S to this state. If it is not better, leave S alone.
c. If the S is better than the current state, then set current state to S.

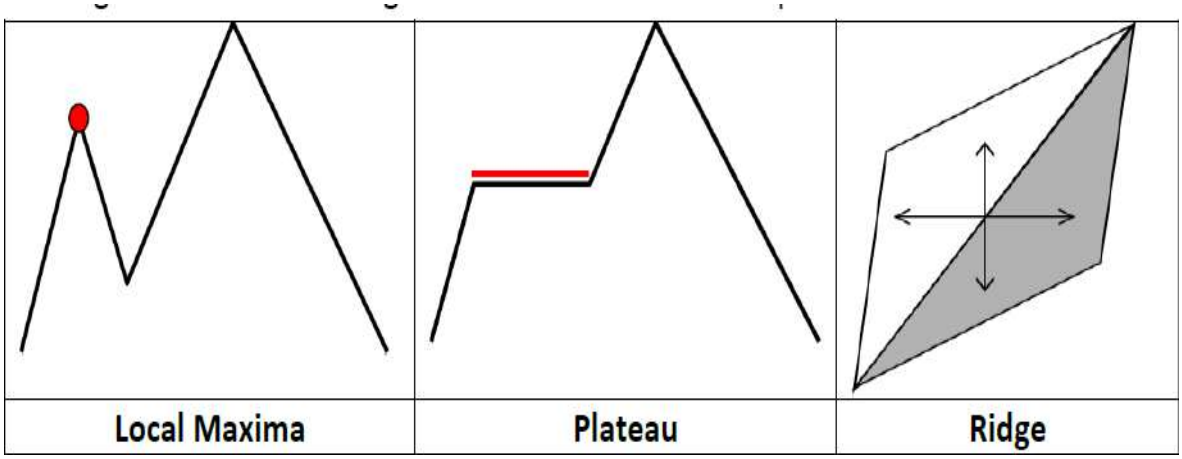
Hill Climbing has three well-known drawbacks:

Sometimes this procedure may lead to a position, which is not a solution, but from which there is no move that improves things. This will happen if we have reached one of the following three states.

(a) A **"local maximum"** which is a state better than all its neighbors, but is not better than some other states farther away. Local maxim sometimes occur within sight of a solution. In such cases they are called "Foothills".

(b) A **"plateau"** which is a flat area of the search space, in which neighboring states have the same value [a plateau is a flat area of the search space in which, a whole set of neighboring states have the same values]. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.

(c) A **"ridge"** which is an area in the search that is higher than the surrounding areas, but cannot be searched in a simple move. It is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has slop.



In each of the previous cases (local maxima, plateaus & ridge), the algorithm reaches a point at which no progress is being made.

To overcome these problems we can

(a) Back track to some earlier nodes and try a different direction. This is a good way of dealing with local maxim.

(b) Make a big jump at some direction to a new area in the search. This can be done by applying two more rules of the same rule several times, before testing. This is a good strategy is dealing with plate and ridges.

(c) Moving in several directions at once.

Hill climbing becomes inefficient in large problem spaces, and when combinatorial explosion occurs. But it is a useful when combined with other methods.

Best First Search (Informed Search)

Best-first search in its most general form is a simple heuristic search algorithm. “Heuristic” here refers to a general problem-solving rule or set of rules that do not guarantee the best solution or even any solution, but serves as a useful guide for problem-solving. Best-first search is a graph-based search algorithm (Dechter and Pearl, 1985), meaning that the search space can be represented as a series of nodes connected by paths.

Best-first search is an algorithm that traverses a graph in search of one or more goal nodes. The defining characteristic of this search is that, unlike DFS or BFS (which blindly examines/expands a cell without knowing anything about it or its properties), best-first search uses an evaluation function (sometimes called a "heuristic") to determine which object is the most promising, and then examines this object. This "best first" behavior is implemented with a Priority Queue.

- DFS is good because it allows a solution to be found without expanding all competing branches. BFS is good because it does not get trapped on dead end paths.
- Best first search combines the advantages of both DFS and BFS into a single method.
- One way of combining BFS and DFS is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does.

OR Graphs

- It is sometimes important to search graphs so that duplicate paths will not be pursued.
- An algorithm to do this will operate by searching a directed graph in which each node represents a point in problem space.
- Each node will contain:
  - ✓ Description of problem state it represents
  - ✓ Indication of how promising it is
  - ✓ Parent link that points back to the best node from which it came
  - ✓ List of nodes that were generated from it
- Parent link will make it possible to recover the path to the goal, once the goal is found.
- The list of successors will make it possible, if a better path is found to an already existing node, to propagate the improvement down to its successors.
- This is called OR-graph, since each of its branches represents an alternative problem solving path.

Implementation of OR graphs

We need two lists of nodes:

- OPEN – nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined. OPEN is actually a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function.
- CLOSED- nodes that have already been examined. We need to keep these nodes in memory if we want to search a graph rather than a tree, since whenever a new node is

- generated; we need to check whether it has been generated before.

Algorithm: Best First Search

1. Start with OPEN containing just the initial state

2. Until a goal is found or there are no nodes left on OPEN do:

a. Pick the best node on OPEN

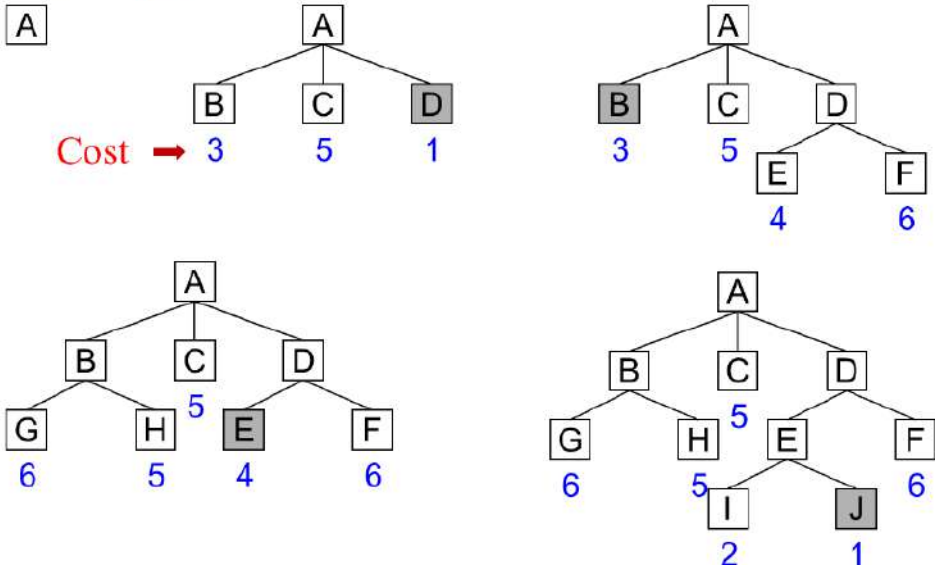
b. Generate its successors

c. For each successor do:

i. If it has not been generated before, evaluate it, add it to OPEN, and record its parent.

ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

Best First Search example



Applications

Best-first search and its more advanced variants have been used in such applications as games and web crawlers.

- In a web crawler, each web page is treated as a node, and all the hyperlinks on the page are treated as unvisited successor nodes. A crawler that uses best-first search generally uses an evaluation function that assigns priority to links based on how closely the contents of their parent page resemble the search query
- In games, best-first search may be used as a path-finding algorithm for game characters. For example, it could be used by an enemy agent to find the location of the player in the game world. Some games divide up the terrain into "tiles" which can either be blocked or unblocked. In such cases, the search algorithm treats each tile as a node, with the neighbouring unblocked tiles being successor nodes, and the goal node being the destination tile.

The A* Algorithm

Best First Search is a simplification of A* Algorithm. The algorithm searches a directed graph in which each node represents a point in the problem space. Each node will contain a description of the problem state it represents and it will have links to its parent nodes and successor nodes. In addition it will also indicate how best it is for the search process. A* algorithm uses have been generated, heuristic functions applied to them, but successors not generated. The list CLOSED contains nodes which have been examined, i.e., their successors generated.

- This algorithm uses following functions:
  1. ***f':*** Heuristic function that estimates the merits of each node we generate. ***f'*** = $g + h'$.
  2. ***f'*** represents an estimate of the cost of getting from the initial state to a goal state along with the path that generated the current node.
  3. ***g:*** The function *g* is a measure of the cost of getting from initial state to the current node.
  4. ***h':*** The function *h'* is an estimate of the additional cost of getting from the current node to a goal state.
  5. The algorithm also uses the lists: OPEN and CLOSED

Algorithm: A*

1. *Start with OPEN containing only initial node. Set that node's g value to 0, its h' value to whatever it is, and its f' value to h'+0 or h'. Set CLOSED to empty list.*
2. *Until a goal node is found, repeat the following procedure: If there are no nodes on OPEN, report failure. Otherwise select the node on OPEN with the lowest f' value. Call it BESTNODE. Remove it from OPEN. Place it in CLOSED. See if the BESTNODE is a goal state. If so exit and report a solution. Otherwise, generate the successors of BESTNODE but do not set the BESTNODE to point to them yet. For each of the SUCCESSOR, do the following:*
   a. *Set SUCCESSOR to point back to BESTNODE. These backwards links will make it possible to recover the path once a solution is found.*
   b. *Compute g(SUCCESSOR) = g(BESTNODE) + the cost of getting from BESTNODE to SUCCESSOR*
   c. *See if SUCCESSOR is the same as any node on OPEN. If so call the node OLD.*
      i. *Check whether it is cheaper to get to OLD via its current parent or to SUCESSOR via BESTNODE by comparing their g values.*
      ii. *If OLD is cheaper, then do nothing. If SUCCESSOR is cheaper then reset OLD's parent link to point to BESTNODE.*
      iii. *Record the new cheaper path in g(OLD) and update f '(OLD).*
   d. *If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so, call the node on CLOSED OLD and add OLD to the list of BESTNODE's successors.*
   e. *If SUCCESSOR was not already on either OPEN or CLOSED, then put it on OPEN and add it to the list of BESTNODE's successors. Compute f'(SUCCESSOR) = g(SUCCESSOR) + h'(SUCCESSOR).*

Observations about A*

- Role of g function: This lets us choose which node to expand next on the basis of not only of how good the node itself looks, but also on the basis of how good the path to the node was.
- h', the distance of a node to the goal. If h' is a perfect estimator of h, then A* will converge immediately to the goal with no search.

Admissibility of A*

- A heuristic function h'(n) is said to be admissible if it never overestimates the cost of getting to a goal state. i.e. if the true minimum cost of getting from node n to a goal state is C then h must satisfy: $h'(n) \leq C$
- If h' is a perfect estimator of h, then A* will converge immediately to the goal state with no search.
- If h' never overestimates h, then A* algorithm is guaranteed to find an optimal path if
- one exists.

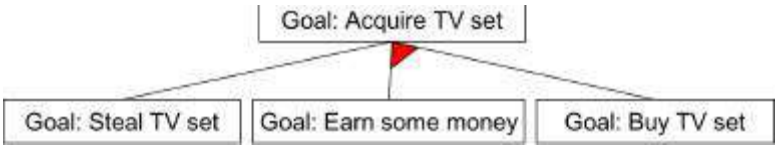
Problem Reduction

AND-OR graphs

- AND-OR graph (or tree) is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.
- This decomposition or reduction generates arcs that we call AND arcs.
- One AND arc may point to any numbers of successor nodes. All of which must then be solved in order for the arc to point solution
- In order to find solution in an AND-OR graph we need an algorithm similar to best – first search but with the ability to handle the AND arcs appropriately.

o ***We define FUTILITY, if the estimated cost of solution becomes greater than the value of FUTILITY then we abandon the search, FUTILITY should be chosen to correspond to a threshold***.

o Following figure shows AND arcs are indicated with a line connection all the components.



The AO* Algorithm

- Rather than the two lists, OPEN and CLOSED, that were used in the A* algorithm, the AO* algorithm will use a single structure GRAPH, representing the part of the search graph that has been explicitly generated so far.
- Each node in the graph will point both down to its immediate successors and up to its immediate predecessors.
- Each node in the graph will also have associated with it an *h'* value, an estimate of the cost of a path from itself to a set of solution nodes.
- We will not store *g* (the cost of getting from the start node to the current node) as we did in the A* algorithm.

- And such a value is not necessary because of the top-down traversing of the edge which guarantees that only nodes that are on the best path will ever be considered for expansion.

Algorithm: AO*

1. Let GRAPH consist only of the node representing the initial state. Call this node INIT, Compute VINIT.

2. Until INIT is labeled SOLVED or until INIT's h' value becomes greater than FUTILITY, repeat the following procedure:

a. Trace the labeled arcs from INIT and select for expansion one of the as yet unexpanded nodes that occurs on this path. Call the selected node NODE.

b. Generate the successors of NODE. If there are none, then assign FUTILITY as the h' value of NODE. This is equivalent to saying that NODE is not solvable. If there are successors, then for each one (called SUCCESSOR) that is not also an ancestor of NODE do the following:

i. Add SUCCESSOR to GRAPH

ii. If SUCCESSOR is a terminal node, label it SOLVED and assign it an h' value of 0

iii. If SUCCESSOR is not a terminal node, compute its h' value

c. Propagate the newly discovered information up the graph by doing the following: Let S be a set of nodes that have been labeled SOLVED or whose h' values have been changed and so need to have values propagated back to their parents. Initialize 5 to NODE. Until S is empty, repeat the, following procedure:

i. If possible, select from S a node none of whose descendants in GRAPH occurs in S. If there is no such node, select any node from S. Call this node CURRENT, and remove it from S.

ii. Compute the cost of each of the arcs emerging from CURRENT. The cost of each arc is equal to the sum of the h' values of each of the nodes at the end of the arc plus whatever the cost of the arc itself is. Assign as CURRENT'S new h' value the minimum of the costs just computed for the arcs emerging from it.

iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost as computed in the previous step.

iv. Mark CURRENT SOLVED if all of the nodes connected to it through the new labeled arc have been labeled SOLVED.

v. If CURRENT has been labeled SOLVED or if the cost of CURRENT was just changed, then its new status must be propagated back up the graph. So add all of the ancestors of CURRENT to S.

VI. Constraint Satisfaction

- Constraint satisfaction is a search procedure that operates in a space of constraint sets. The initial state contains the constraints that are originally given in the problem description.
- A goal state is any state that has been constrained "enough" where "enough" must be defined for each problem.
- For example, in cryptarithmetic problems, enough means that each letter has been assigned a unique numeric value.
- Constraint Satisfaction problems in AI have goal of discovering some problem state that satisfies a given set of constraints.
- Design tasks can be viewed as constraint satisfaction problems in which a design must be created within fixed limits on time, cost, and materials.
- Constraint Satisfaction is a two-step process:
    1. First constraints are discovered and propagated as far as possible throughout the system.
    2. Then if there is still not a solution, search begins. A guess about something is made and added as a new constraint.

Example: Cryptarithmetic Problem

Constraints:

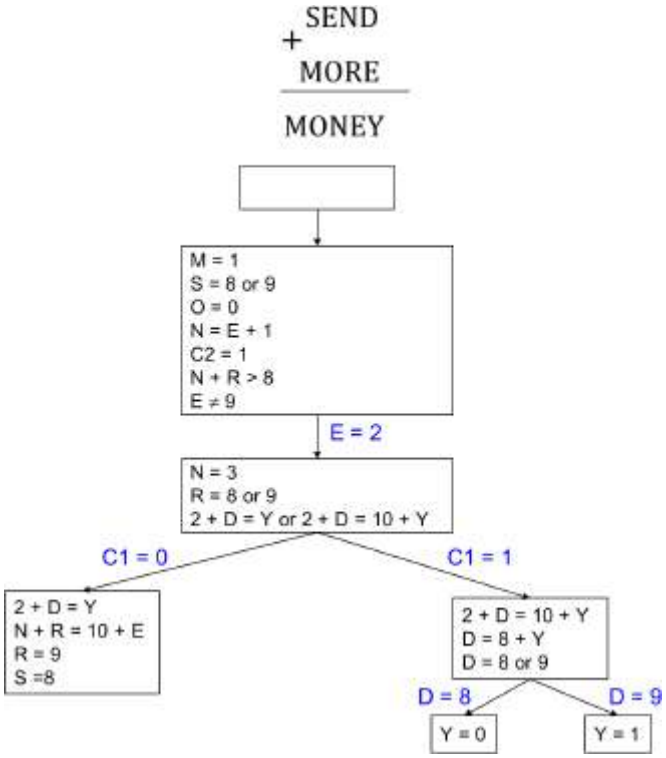
• No two letters have the same value

• The sums of the digits must be as shown in the problem

Goal State:

• All letters have been assigned a digit in such a way that all the initial constraints are satisfied

Input State



The solution process proceeds in cycles. At each cycle, two significant things are done:

1. Constraints are propagated by using rules that correspond to the properties of arithmetic.
2. A value is guessed for some letter whose value is not yet determined.

Solution:

```
  9567
+ 1085
--------
 10652
```

Algorithm: Constraint Satisfaction

1. Propagate available constraints. To do this first set OPEN to set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until OPEN is empty:

    a. Select an object OB from OPEN. Strengthen as much as possible the set of constraints that apply to OB.
    b. If this set is different from the set that was assigned the last time OB was examined or if this is the first time OB has been examined, then add to OPEN all objects that share any constraints with OB.
    c. Remove OB from OPEN.

2. If the union of the constraints discovered above defines a solution, then quit and report the solution.

3. If the union of the constraints discovered above defines a contradiction, then return the failure.

4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this loop until a solution is found or all possible solutions have been eliminated:
    a. Select an object whose value is not yet determined and select a wayof strengthening the constraints on that object.
    b. Recursively invoke constraint satisfaction with the current set of constraints augmented by strengthening constraint just selected.

VII. Means-Ends Analysis

Most of the search strategies either reason forward of backward however, often a mixture of the two directions is appropriate. Such mixed strategy would make it possible to solve the major parts of problem first and solve the smaller problems that arise when combining them together. Such a technique is called "Means - Ends Analysis".

The means -ends analysis process centres around finding the difference between current state and goal state. The problem space of means - ends analysis has an initial state and one or more goal state, a set of operate with a set of preconditions their application and difference functions that computes the difference between two state a(i) and s(j). A problem is solved using means - ends analysis by

1. Computing the current state s1 to a goal state s2 and computing their difference D12.

2. Satisfy the preconditions for some recommended operator op is selected, then to reduce the difference D12.

3. The operator OP is applied if possible. If not the current state is solved a goal is created and means- ends analysis is applied recursively to reduce the sub goal.

4. If the sub goal is solved state is restored and work resumed on the original problem.

(the first AI program to use means - ends analysis was the GPS General problem solver)

Means- ends analysis is useful for many human planning activities. Consider the example of planning for an office worker. Suppose we have a different table of three rules:

1. If in our current state we are hungry, and in our goal state we are not hungry, then either the "visit hotel" or "visit Canteen" operator is recommended.

2. If our current state we do not have money, and if in your goal state we have money, then the "Visit our bank" operator or the "Visit secretary" operator is recommended.

3. If our current state we do not know where something is , need in our goal state we do know, then either the "visit office enquiry" , "visit secretary" or "visit co-worker " operator is recommended.

Algorithm: Means-Ends Analysis

1. Compare CURRENT to GOAL. If there are no differences between them then return.
2. Otherwise, select the most important difference and reduce it by doing the following until success or failure is signaled:

> *a. Select an as yet untried operator O that is applicable to the current difference. If there are no such operators, then signal failure.*
>
> *b. Attempt to apply O to CURRENT. Generate descriptions of two states: OSTART, a state in which O's preconditions are satisfied and O-RESULT, the state that would result if O were applied in O-START.*
>
> *c. If*
> *(FIRST-PART ← MEA(CURRENT, O-START))*
> *and*
> *(LAST-PART ← MEA(O-RESULT, GOAL))*
> *are successful, then signal success and return the result of concatenating FIRST-PART, O, and LAST-PART.*

INTELLIGENT AGENTS

For an increasingly large number of applications, we require systems that can *decide for themselves* what they need to do in order to satisfy their design objectives. Such computer systems are known as *agents*. Agents that must operate robustly in rapidly changing, unpredictable, or open environments, where there is a significant possibility that actions can *fail* are known as *intelligent agents*, or sometimes *autonomous agents*. An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents.

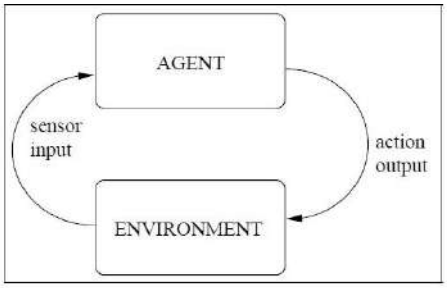
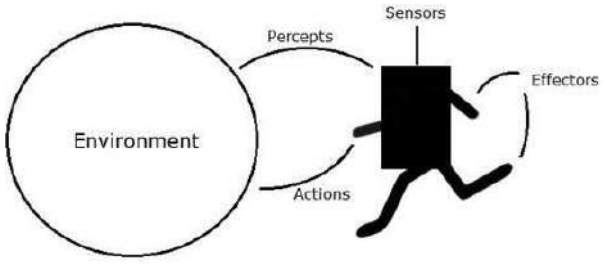


Figure 1.1 An agent in its environment. The agent takes sensory input from the environment, and produces as output actions that affect it. The interaction is usually an ongoing, non-terminating one.

What are Agent and Environment?

An **agent** is anything that can perceive its environment through **sensors** and acts upon that environment through **effectors.**

- A **human agent** has sensory organs such as eyes, ears, nose, tongue and skin parallel to the sensors, and other organs such as hands, legs, mouth, for effectors.
- A **robotic agent** replaces cameras and infrared range finders for the sensors, and various motors and actuators for effectors.
- A **software agent** has encoded bit strings as its programs and actions.



An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives.

An intelligent agent is one that is capable of flexible autonomous action in order to meet its design objectives. The Flexibility includes:

- reactivity: intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives;
- pro-activeness: intelligent agents are able to exhibit goal-directed behaviour by taking the initiative in order to satisfy their design objectives;
- Social ability: intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

Agent Terminology

- **Performance Measure of Agent** − It is the criteria, which determines how successful an agent is.
- **Behavior of Agent** − It is the action that agent performs after any given sequence of percepts.
- **Percept** − It is agent's perceptual inputs at a given instance.
- **Percept Sequence** − It is the history of all that an agent has perceived till date.
- **Agent Function** − It is a map from the precept sequence to an action.

Rationality

Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.

Rationality is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of rationality.

What is Ideal Rational Agent?

An ideal rational agent is the one, which is capable of doing expected actions to maximize its performance measure, on the basis of −

- Its percept sequence
- Its built-in knowledge base

Rationality of an agent depends on the following four factors −

- The **performance measures**, which determine the degree of success.
- Agent's **Percept Sequence** till now.
- The agent's **prior knowledge about the environment**.
- The **actions** that the agent can carry out.

A rational agent always performs right action, where the right action means the action that causes the agent to be most successful in the given percept sequence. The problem the agent solves is characterized by Performance Measure, Environment, Actuators, and Sensors (PEAS).

The Structure of Intelligent Agents

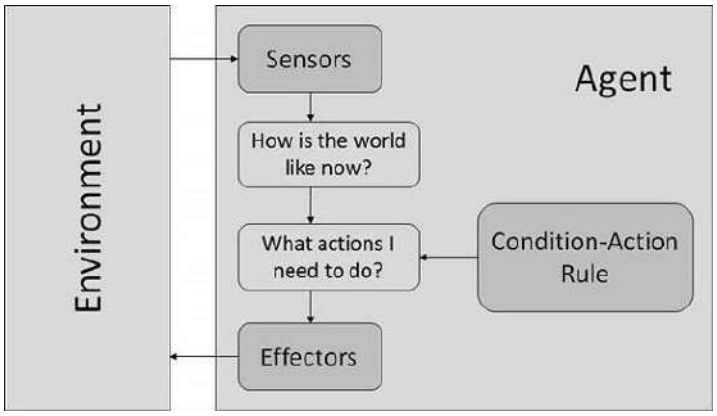
Agent's structure can be viewed as −

- Agent = Architecture + Agent Program
- Architecture = the machinery that an agent executes on.
- Agent Program = an implementation of an agent function.

Simple Reflex Agents

- They choose actions only based on the current percept.
- They are rational only if a correct decision is made only on the basis of current precept.
- Their environment is completely observable.

Condition-Action Rule − It is a rule that maps a state (condition) to an action.



Model Based Reflex Agents

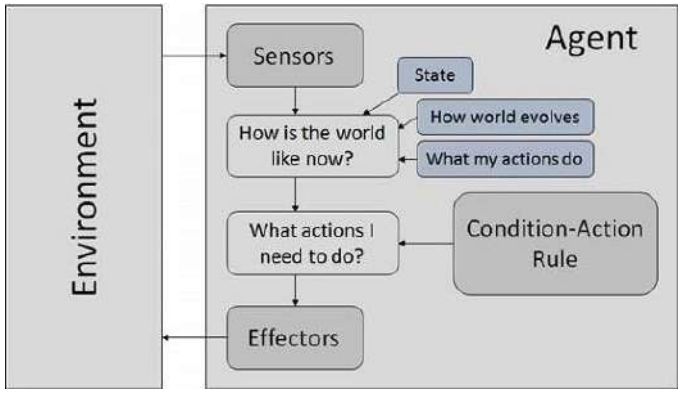
They use a model of the world to choose their actions. They maintain an internal state.

Model − The knowledge about "how the things happen in the world".

Internal State − It is a representation of unobserved aspects of current state depending on percept history.

Updating the state requires the information about −

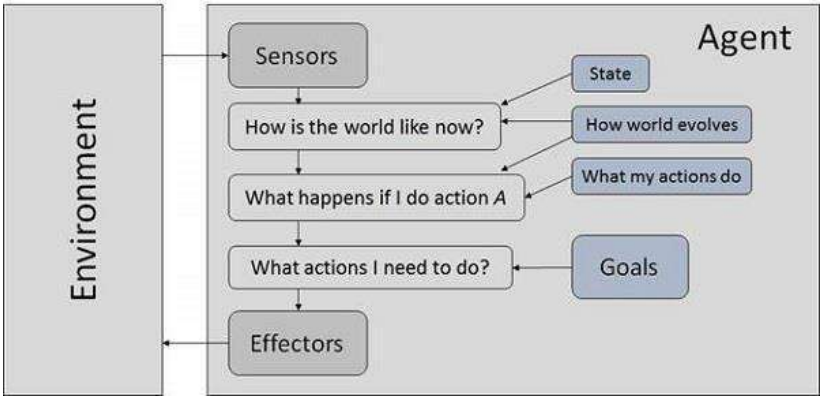
- How the world evolves.
- How the agent's actions affect the world.



Goal Based Agents

They choose their actions in order to achieve goals. Goal-based approach is more flexible than reflex agent since the knowledge supporting a decision is explicitly modeled, thereby allowing for modifications.

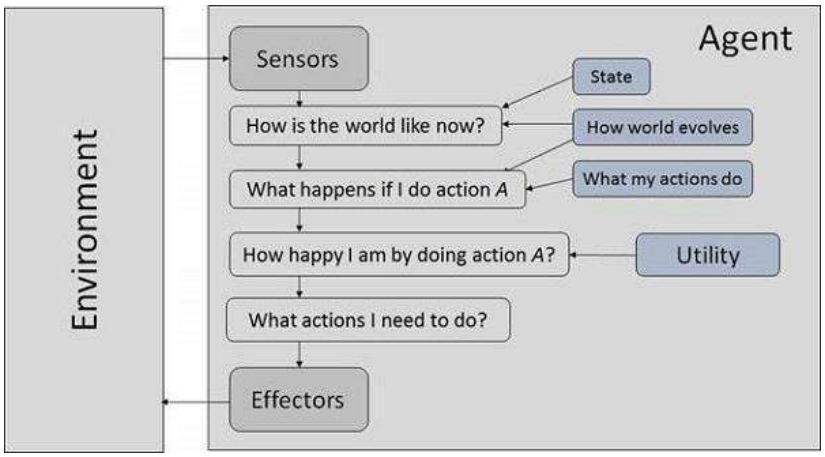
Goal − It is the description of desirable situations.



Utility Based Agents

They choose actions based on a preference (utility) for each state. Goals are inadequate when –

- There are conflicting goals, out of which only few can be achieved.
- Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.
-



Nature of Environments

Some programs operate in the entirely **artificial environment** confined to keyboard input, database, computer file systems and character output on a screen.

In contrast, some software agents (software robots or softbots) exist in rich, unlimited softbots domains. The simulator has a **very detailed, complex environment**. The software agent needs to choose from a long array of actions in real time. A softbot designed to scan the online preferences of the customer and show interesting items to the customer works in the **real** as well as an **artificial** environment.

The most famous **artificial environment** is the **Turing Test environment**, in which one real and other artificial agents are tested on equal ground. This is a very challenging environment as it is highly difficult for a software agent to perform as well as a human.

Turing Test

- The success of an intelligent behavior of a system can be measured with Turing Test.

- Two persons and a machine to be evaluated participate in the test. Out of the two persons, one plays the role of the tester. Each of them sits in different rooms. The tester is unaware of who is machine and who is a human. He interrogates the questions by typing and sending them to both intelligences, to which he receives typed responses.

- This test aims at fooling the tester. If the tester fails to determine machine's response from the human response, then the machine is said to be intelligent.

Properties of Environment

The environment has multifold properties –

- **Discrete / Continuous** − If there are a limited number of distinct, clearly defined, states of the environment, the environment is discrete (For example, chess); otherwise it is continuous (For example, driving).

- **Observable / Partially Observable** − If it is possible to determine the complete state of the environment at each time point from the percepts it is observable; otherwise it is only partially observable.

- **Static / Dynamic** − If the environment does not change while an agent is acting, then it is static; otherwise it is dynamic.

- **Single agent / Multiple agents** − The environment may contain other agents which may be of the same or different kind as that of the agent.

- **Accessible / Inaccessible** − If the agent's sensory apparatus can have access to the complete state of the environment, then the environment is accessible to that agent.

- **Deterministic / Non-deterministic** − If the next state of the environment is completely determined by the current state and the actions of the agent, then the environment is deterministic; otherwise it is non-deterministic.

- **Episodic / Non-episodic** − In an episodic environment, each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself. Subsequent episodes do not depend on the actions in the previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.